VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2011 **Janusz Supik**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Kódování mpeg streamu algoritmem CSA
Coding Mpeg Stream Using CSA Algorithm

2011 **Janusz Supik**

List zadání

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne 3.5.2011 . . . . . .

podpis

Poděkování

Touto cestou bych rád poděkoval především **Ing. Davidu Seidlovi** za odborné vedení, neocenitelné rady a připomínky k vypracování této bakalářské práce.

Abstrakt

Tato bakalářská práce se soustředí na vývoj jednoduchého a nenáročného softwaru pro kódování MPEG transportního streamu. Nemusí jít nutně o kódování na straně velkého poskytovatele televizních služeb, ale spíše o kódování v rámci lokálních sítí. Program by měl umožňovat jednoduchou konfiguraci a neměl by příliš zatěžovat systém na kterém pracuje. Pro funkční vývoj je také nutná znalost formátu vysílání, hlavně přenášených paketů. Teoretická část se bude zaměřovat na samotný transportní stream - způsob přenášení dat, spojování paketů a rozbor transportních paketů. Důležitou části práce, která se zabývá programem, je porovnání výkonu s již existujícími nástroji pro kódování.

Klíčová slova: transportní paket; transportní stream; adaptační pole; payload; záhlaví; DVB; CSA; klasická a paralelní implementace

Abstract

This bachelor thesis is focused on development simple and undemanding software for scrambling of MPEG transport stream. It does not have to be focused to scrambling on the side of provider of TV service, but rather on small local networks. Program should provide easy configuration and shoudln´t burden system. For developments is necessary to know broadcast format, especially transport packets. Theoretical part will be focused on transport stream - broadcast methods, packet merging and analyze of transport packets. Important part of work, which deals with program, is comparison with existing tools for decrypting.

Keywords: transport packet; transport stream; adaptation field; payload; header; DVB; CSA; classic and parallel implementation

Seznam použitých zkratek

API - Application Programming Interface
CSA - Common Scrambling Algorithm
DVB - Digital Video Broadcasting
MPEG - Moving Picture Experts Group
MPEG TS - MPEG Transport Stream
PAT - Program Association Table
PES - Packetized Elementary Stream
PID - Packet IDentification
PMT - Program Map Table
UDP - User Datagram Protocol
VLC - VideoLan Client

Obsah

1. Úvod

Digitální televize. Pojem, který v poslední době slýcháme stále častěji. A to nejen kvůli digitalizace TV vysílání v České republice. DVB vysílání má v dnešní době mnoho podob a specifikací. Mezi nejznámější patří: DVB-C, DVB-S, DVB-T, DVB-IPTV pro přenos televize prostřednictvím počítačových sítí. V neposlední řadě DVB-H pro přenosná zařízení jako jsou např.: mobilní telefony, PDA a jiná. Jelikož na světě nežijí jen poctiví lidé, kteří nesledují vysílání, které není pro ně určeno, vzniká potřeba tato vysílání kódovat.

Existují programy umožňující kódování streamu jako VLC Media Player, ale tyto programy jsou velice náročné na paměť jelikož neposkytují pouze tuto službu. Mým cílem bylo vytvořit malou a nenáročnou aplikaci pro operační systém Linux, která bude kódování realizovat a to pomocí DVB - CSA. Aplikaci budu psát v programovacím jazyce C++. Rozhodl jsem se pro tento jazyk z důvodu předchozích studijních zkušeností. Rozhodoval jsem se mezi C++ a Javou. Javu jsem si nevybral z důvodu, že s ní pod Linuxem nemám žádnou osobní zkušenost a navíc je Java pomalejší oproti C++.

Jako první se budu zabývat spíše teoretickou stránkou problému. Rozeberu principy fungování a vysílání digitálního televizního signálu. Na začátku popíšu MPEG transportní stream. Následně zanalyzuji formát přenášených transportních paketů. Poté bych se chtěl stručně zabývat algoritmem CSA (popsat vnitřní uspořádaní HW realizace tohoto algoritmu). A na konci kapitoly stručně rozeberu celkový princip DVB a jednotlivé DVB specifikace.

V následující kapitole budu popisovat samotnou implementaci a aktuální stav programu. Po tomto stručném popisu připojím analýza řešení, charakteristiku CSA knihovny z distribuce projektu VideoLan, popis dvou verzí implementace, pomocí sériové a paralelní verze CSA. Na závěr uvedu ukázky funkčnosti - výpisy přijatých a odeslaných dat a srovnání výkonu mého programu a již existujících programů, které kódování umožňují.

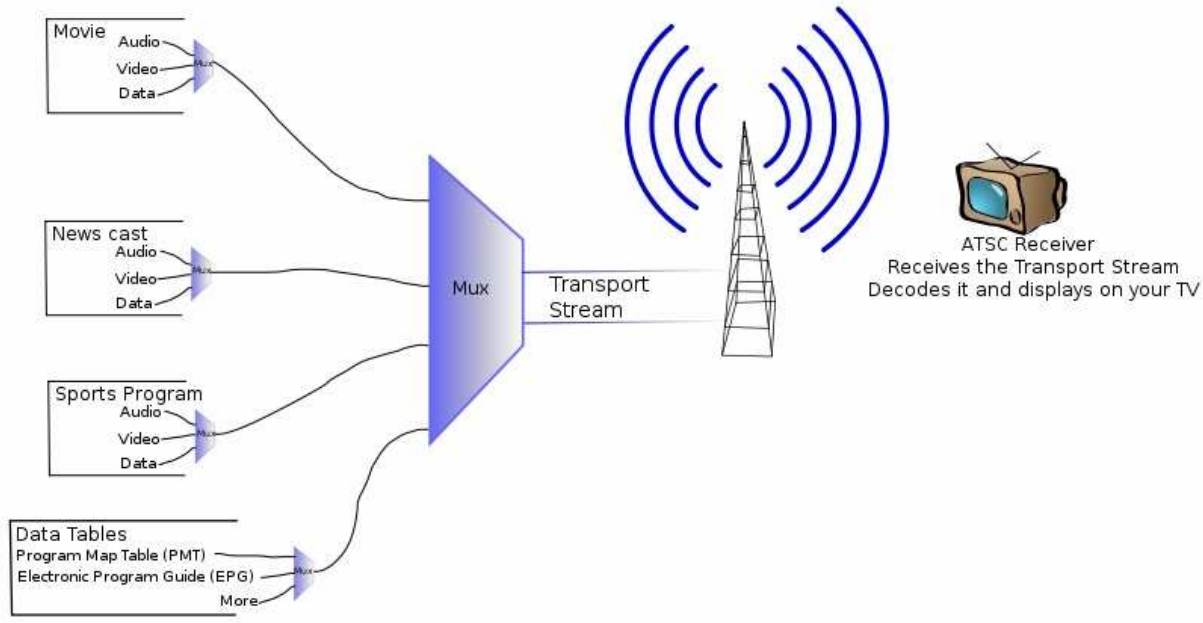
2. MPEG Transport stream a jeho použití v DVB

2.1. Transport stream

MPEG TS je formát pro posílání nebo ukládání (soubory s příponou .TS) zvuku, videa a řídících dat. Je navržen pro posílání dat v reálném čase prostřednictvím nespolehlivých transportních médií. Specifikuje formát zapouzdření dat za účelem detekce chyb, synchronizace a dodržení integrity dat. Má dvě hlavní použití: posílání pomocí počítačové sítě nebo využití v digitálním televizním vysílání. Digitální televizní vysílání je nejrozšířenější místo uplatnění transportního streamu (DVB nebo ATSC).

V případě digitálního vysílání je transportní stream multiplexem několika kanálů (většinou digitálních televizních programů) do jednoho proudu dat, za účelem efektivnějšího využití přenosového média. Jeden datový tok znamená pouze jedny řídící data, jednu nosnou frekvenci. MPEG Transport Stream je také nazýván Multi-program TS.

2.1.1. Multiplex



Obrázek 1. [Zdroj: www.wikipedia.org]

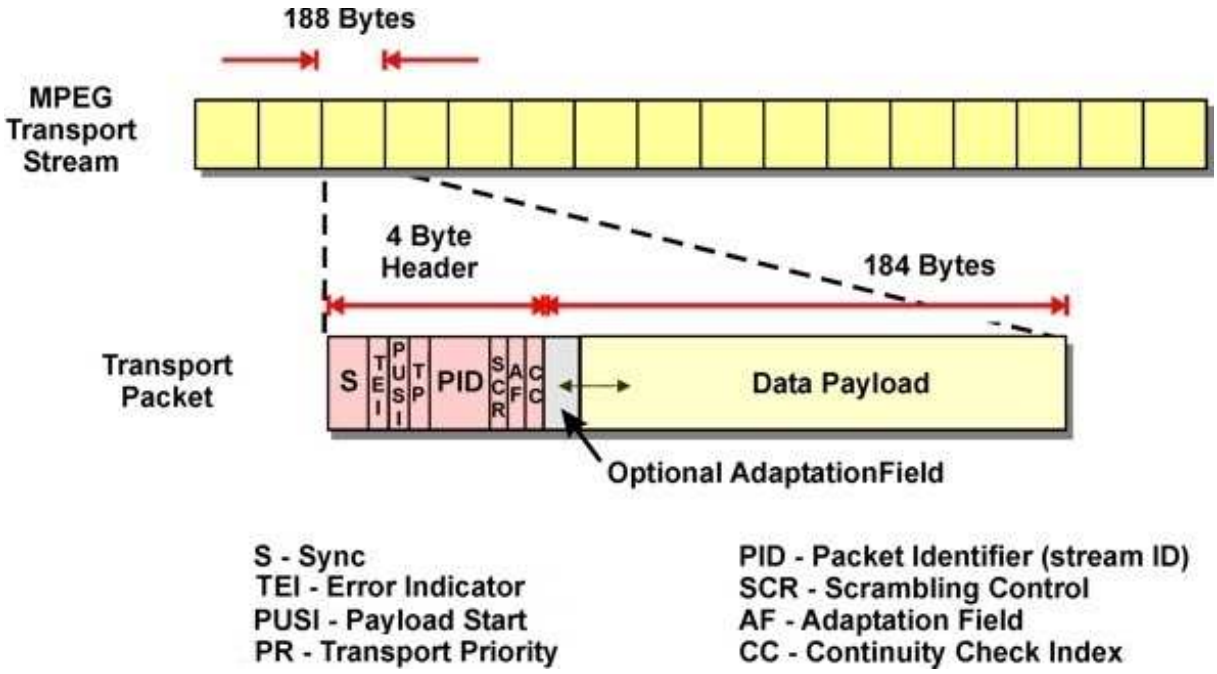
Multiplex je jednou z nejdůležitějších částí TS. Cituji z literatury pana Legíňa: "*Komprimované datové toky komprimovaného obrazového a zvukového signálu (zdrojové signály obrazu a zvuku) nepřicházejí do hlavního multiplexeru v dlouhém souvislém sledu. Jsou rozděleny na menší jednotky tzv. pakety, které jsou opatřeny informačním záhlavím. Toto rozdělení umožňuje vzájemnou synchronizaci obrazu, zvuku i jiných přídavných dat v dekodéru,*

protože periodická struktura paketu vykazuje určité záchytné body dané informacemi v záhlaví. Vytvářením paketů se dílčí signály multiplexují do výsledného toku. Ten může obsahovat nejenom jeden televizní program doprovázený v téže časové souvislosti (základně) zvukem (jedním i několikajazyčným) a přídavnými daty (teletext, informačními tabulkami), ale může být v hlavním multiplexu spojováno několik televizních programů navzájem časově nezávislých.

Na obrázku 1 je znázorněno vytvoření transportního toku pomocí multiplexoru. Z důvodu výskytu rušení při přenosu (na typu přenosu nezáleží - rušení je všude) a při kanálovém kódování (před modulací nosné vlny) je potřeba signál zabezpečit přídavnými ochranami. Pro tento dálkový přenos se hodí krátké a stejně dlouhé pakety, které mimo to usnadňují synchronizaci jednotlivých částí signálu v dekodéru.

Přenos televizního signálu po částech, tj. v paketech, představuje velkou schopnost různého zpracování (flexibilitu). Tyto kratší celky se snadno uchovávají v paměti a je možné je skládat do různých podob výsledného toku pří různých přenosových cestách. Dekodér přijímače pak podle povelů v záhlavích, tj. dekódovacích a prezentačních značek, vybírá pro dekódování a ve vhodné časové souvislosti reprodukuje sobě příslušející části jednotlivých programů"[1].

2.1.2. Formát přenášených dat



Obrázek 2. [Zdroj: www.iptvdictionary.com]

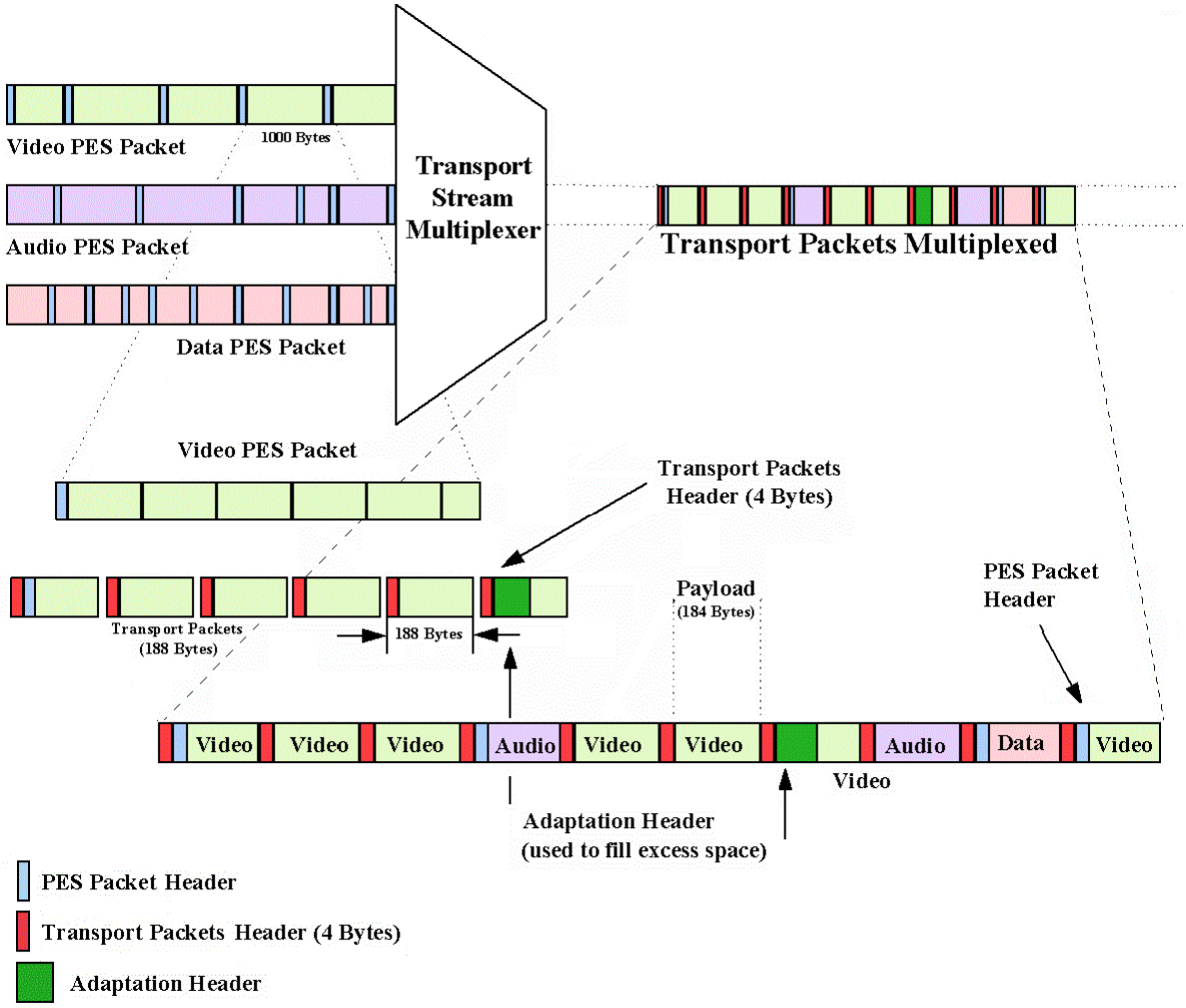
Paket je základní přenosovou jednotkou v transportních streamech. Přenáší se v blocích po sedmi paketech. Z důvodů rušení na všech částech cesty paketu k cílovému přijímači se dlouhé pakety elementárních toků transformují na krátké pakety stejné délky. Ty je pak jednodušší zabezpečit proti chybám vznikajícím u kanálového kódování. Paket má celkovou

délku 188 bytů. Čtyři první byty jsou vyhrazeny pro záhlaví, které nese informace o daném paketu. Užitečným datům je vyhrazeno 184 bytů. Doporučený počet bytů užitečných dat je násobkem čísla 8 a to z důvodu, že po každém sledu osmi bytů probíhá scramblování dat.

Na obrázku číslo 2 můžete vidět záhlaví standardního paketu transport streamu. Je rozděleno do osmi skupin, můžeme zde rovněž vidět náznak významu jednotlivých skupin. Význam jednotlivých skupin:

- **SYNC - SYNC BYTE -** Synchronizační byte. Oznamuje začátek paketu. Jeho hodnota je 0x47.
- **TEI - Transport Error Indicator** anebo E**rror Indicator**. Indikátor chybného přenosu. Nastavuje ho demodulátor, když nemůže opravit chyby způsobené přenosem. Oznamuje demultiplexeru neopravitelnou chybu.
- **PUSI - Payload Unit Start Indicator -** udává, že se v daném paketu vyskytuje záhlaví paketu elementárního toku (PES - Paketový elementární stream) nebo programové informační tabulky a jejich parametry.
- **TP - Transport priority -** vyznačuje, které pakety se stejným PID se při přetížení přenosové cesty mají přenášet přednostně.
- **PID - Packet ID -** třinácti bitové číslo. Shodná hodnota pro pakety příslušející k sobě v elementárním streamu (obrazové, zvukové a datové).
- **SCR - Scrambling Control -** někdy taky označovaný jako **TSC** - Transport Scrambling Control. Označuje zda je daný paket kódován. Dvou bitové pole nabývající hodnot: **00** - nekódováno; **01 -** nekódováno, může být použito jako libovolný příznak definovaný poskytovatelem služby; **10 -** data kódována sudým klíčem; **11 -** data kódovaná lichým klíčem.
- **AF - Adaptation Field Control -** udává informaci o podílu adaptačního pole v datové části paketu. Jeho velikost je 2 bity. Nabývá hodnot: **00 -** může být použito jako libovolný příznak definovaný poskytovatelem služby; **01 -** adaptační pole v daném paketu není, paket obsahuje pouze užitečná data; **10 -** paket obsahuje pouze adaptační pole a žádná užitečná data; **11 -** v paketu je adaptační pole následováno užitečnými daty;
- **CC - Continuity Counter -** čítá pakety se stejným PID. Tím může odhalit ztrátu paketu nebo jejich špatné pořadí.

Jelikož většina paketů elementárního streamu je přenášená ve více paketech transportního streamu, tudíž většina paketů TS přenáší pouze data PES. Když je přenášeno záhlaví paketu PES je nastaven příznakový bit PUSI na hodnotu 1, z toho vyplývá, že v každém TS paketu může začínat maximálně 1 PES paket. Na obrázku 3 je naznačeno rozdělení PES paketu do paketů transportního streamu.



Obrázek 3.[Zdroj: www.une.edu.ve]

2.1.3. Adaptační pole

V předchozím odstavci padly zmínky o adaptačním poli, ale nikde sem nevysvětlil co, to vlastně je. Jeho přítomnost je indikována příznakovými bity AFC v záhlaví paketu transportního streamu. Jelikož délka paketu elementárního streamu (PES) může být různá, maximálně však 65 526, nemusí se pokaždé rozdělit bezezbytku do 184 bytových paketů transportního streamu. Jelikož tyto pakety mají předepsanou délku na 188 bytů (4 byty záhlaví a 184 užitečná data) bytů, je nutno tyto chybějící data doplnit tak, aby paket měl danou délku. Tato doplňující data se nacházejí v již zmiňovaném adaptačním poli. Toto pole se nachází v paketech přímo za záhlavím a až za tímto polem se nachází užitečná data (payload).

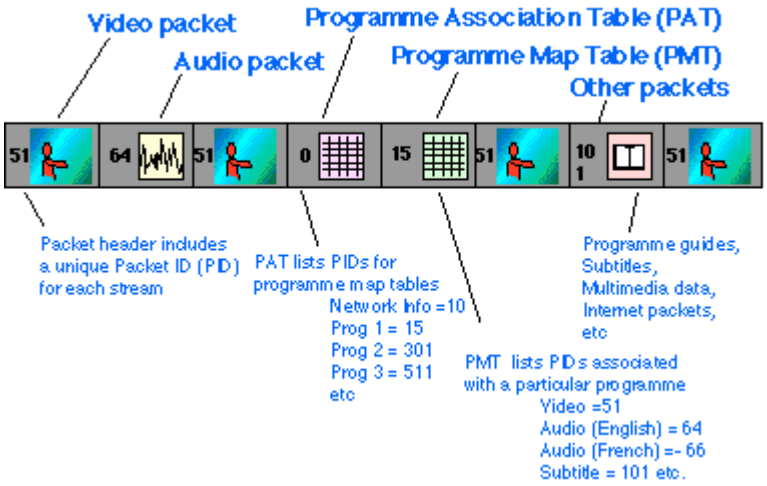
V prvním bytu pole se nachází informace o jeho délce. Dále obsahuje návěstí a informace závislé na těchto návěstích. Následuje počet bytů, který doplní data PES paketu do délky 184. V případě MPEG-2 tyto doplňující byty mají hodnotu 0xFF.

Nevyskytuje se u všech paketů, ale musí se vyskytovat alespoň co 0,1 s a to z důvodu, že obsahuje taky důležité řídící informace pro rekonstrukci obrazu a zvuku v dekodéru. Nejdůležitější je bezesporu navěstí PCR (Programme Clock Reference) - programové referenční hodinové impulzy. Ty mají za úkol synchronizovat v dekodéru zdroj hodinových impulzů.

2.1.4. Tabulky transportního streamu

Na přijímací straně přichází do dekodéru již signál po demodulaci a odstranění chyb zapříčiněných přenosem. Dekodér podle údajů v záhlaví paketu, přesněji podle PID určí, které pakety přísluší přehrávanému kanálu. Z důvodu synchronizace obrazu a zvuku daného kanálu se nejméně jednou za 0,1 s z adaptačního pole získá referenční hodnota hodinových impulzů PCR (Program Clock Reference). Touto hodnotou se synchronizuje generátor pravidelných hodinových impulzů na frekvenci 27 MHz (STC- System Time Clock). Synchronizační impulzy se porovnají s impulzy STC a podle jejich rozdílu se oscilátor doladí. Když už je dekodér synchronizován, musí znát momenty kdy má dané pakety z paměti načíst a použít (zobrazit anebo přehrát). K tomu slouží návěstí PTS a DTS, PTS (Presentation Time Stamp), které udává kdy se má daný paket prezentovat a DTS (Decoding Time Stamp) označuje začátek dekódování.

Jelikož má transportní stream strukturu multi-programového toku, musí být jednotlivé programy popsány v tabulce PMT (Program Map Table) a jednotlivé elementární toky tohoto kanálu mají své PID zaznamenány v této tabulce. Předpokládejme, že v jednom streamu se vyskytují 4 kanály. Každý kanál se skládá z jednoho elementárního video streamu a jednoho nebo dvou zvukových streamů a potřebných řídících dat. Přijímač musí dekódovat pouze užitečná data paketů náležících tomuto kanálu. Zbytek dat může dekodér zahodit.



Obrázek 4. [Zdroj: http://knol.google.com/k/mpeg-2-transmission]

Obrázek 4 znázorňuje transportní stream spolu s řídícími tabulkami a stručné schéma struktury dat v nich.

PAT - Při zapnutí dekodéru je třeba určit strukturu multiplexovaného datového toku. To se provádí právě pomocí tabulky PAT (Program Association Table). Ta je přenášená pakety s hodnotou PID 0x0000. Tyto pakety jsou zpracovány jako první. Tím dekodér získá schéma streamu. Dekodér pak pokračuje ve výběru zvoleného programu. Jednotlivé PID pro zvolený kanál nalezneme v tabulce PMT.

PMT - obsahuje informace o programech. Každému programu přináleží právě jedna PMT. Poskytuje informace o každém programu obsaženém v transport streamu, zahrnuje číslo programu a seznam elementárních streamů náležících tomuto programu. Může obsahovat i další volitelné informace pro popis celého programu, stejně jako popis všech jeho elementárních streamů.

2.2. CSA

Common Scrambling Algorithm je používán v DVB systémech pro kódování video stremů. CSA bylo specifikováno organizací ETSI (European Telecommunication Standards Institute). V roce 1994 byl přejat pro standardy DVB. Jako první se začal používat v kabelových systémech. Byl šířen k uživatelům ve formě hardwarového čipu. Ačkoli nějaké informace o implementaci ve vyšších programovacích jazycích byly specifikovány v patentu, tak algoritmus nebyl nikdy oficiálně zveřejněn. V roce 2002 byl program impelemntující algoritmus zveřejněn v binární podobě, zanedlouho potom byl převeden zpětně do podoby vyššího programovacího jazyku.

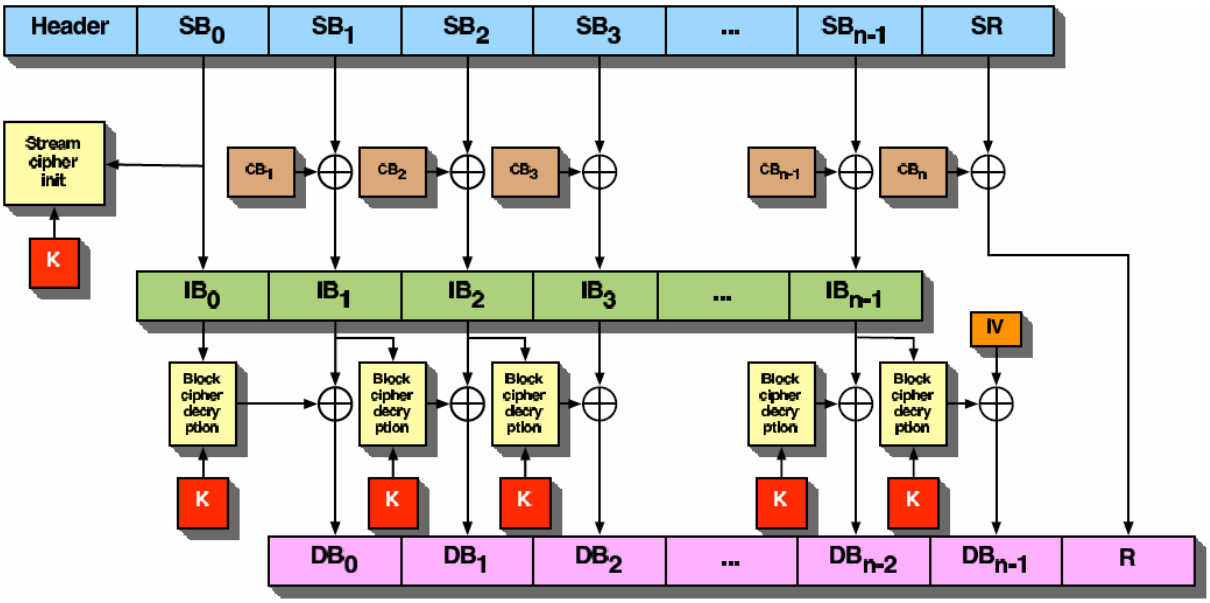
CSA algoritmus může být považován za aplikaci složenou ze 2 vrstev kódování: vrstvu blokového kódování a vrstvu streamového kódování. Jako první při kódování je aplikováno blokové kódování a následně je použito kódování streamové. Tento postup bývá nazýván kaskádovým kódováním. Při dekódování je postup přesně opačný, jako první se dekóduje kódování streamové a pak blokové. Oba typy kódování jsou inicializovány stejným 64 bitovým řídícím slovem. Tyto řídící slova jsou poskytovány řídícím mechanismem, který je generuje na základě kódovaných řídících zpráv z transport streamu (tam se přenášejí jako řídící data).

2.2.1. Kaskádové kódování

Kódovací (scramble) algoritmus si můžeme představit jako kaskádu blokového a streamové kódování. Stručně popíšu jak je zkombinováno blokové a streamové kódování.

Pro kódování jsou užitečná data paketu rozdělená do bloků (DB) po osmi bytech. Pokud paket obsahuje adaptační pole, existuje malá šance, že délka dat nebude násobkem osmi. Pokud tato situace nastane poslední blok je menší než osm bytů a je nazván zbytkem (residue).

Následující obrázek ukazuje schéma kaskádového dekódování. Kódování probíhá analogicky, ale opačným směrem.



Obrázek 5. [Zdroj: www.cdc.informatik.tu-darmstadt.de]

Sekvence osmi bytových bloků jsou blokově zakódovány v opačném pořadí, než v jakém jsou uspořádány po vytvoření, a zbytek (residue) zůstává v původním stavu. Poslední výstup bloků IB (intermediate blocks - přechodové bloky) je následně použit jako NONCE (number used once - inicializační vektor, který se použije pouze jednou a je unikátní) při streamovém kódování. Prvních 8 bytů Keystreamu (proud pseudonáhodných znaků, použití při vytváření kódovaných zpráv) generovaných pomocí streamového kódování je pomocí logické operace XOR upraveno na zakódované bloky IB_i pro i >= 1 a nakonec se zbytkem (residue).

2.2.2. Blokové kódování

Common Scrambling Algorithm využívá iterované blokové kódování, které pracuje na úrovni bytů na 64 bitových blocích dat a využívá přitom 64 bitový klíč. Každá iterace blokového kódování využívá stejnou transformační funkci, která operuje s osmi bytovým vektorem dat a jedním bytem rozšířeného klíče (expanded key), má na výstupu osmi bytový vektor. Tato transformace je použita 56 krát.

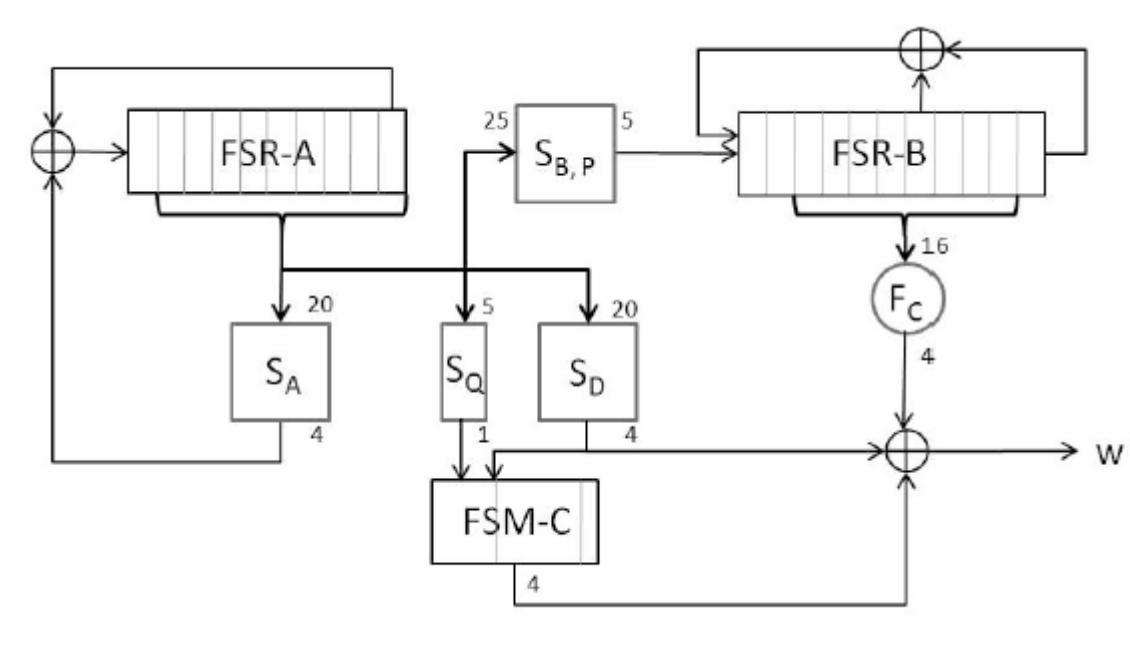
Transformační funkce ϕ je založená na funkcích f a f´. Tyto funkce jsou permutacemi (vzájemně odlišné) na hodnotách všech bytů a můžeme si je představit jako S-Boxy kódování. Obě permutace mají maximální možnou délku cyklu a jsou vzájemně provázané pomocí bitové permutace $f = \sigma \circ f'$.

2.2.3. Streamové kódování

Streamové kódování využívá pro svou činnost dva zpětnovazební posuvné registry FSR- feedback-shift-registers a slučovací obvod (combiner, FSM-C) s vnitřní pamětí. Obecné schéma je znázorněno na obrázku 6.

Oba registry FSR se skládají z deseti segmentů a každá část obsahuje slovo o délce 4 bitů. Combiner se skládá ze dvou segmentů, z nichž každý obsahuje 4 bity. Navíc obsahuje jeden přenosový bit (carry). Celková velikost je tedy 89 bitů.

Během generování keystream-u (série pseudonáhodných znaků, které se zkombinují s původní zprávou pro vytvoření kódované zprávy) je registr FSR-A nezávislý. Funkce pro aktualizace stavů FSR-A není lineární. Pro výpočet dalšího stavu se rovněž využívá výstup s-boxu S_A. FSR-B i FSM-C využívá jako vstup nelineární výstup z FSR-A a to tak, že tento výstup je přiveden do s-boxů S_B, S_D, S_P a S_Q. Výstupy z S_B a S_P jsou využity k modifikování stavů FSR-B a výstupy z S_D a S_Q pro FSM-C. Konkrétní funkce pro aktualizace stavů každé části popíšu detailněji v následující části.



Obrázek 6. [Zdroj: http://eprints.qut.edu.au/]

Když generátoru keystreamu v čase t přijde hodinový impulz tak FSR-A, FSR-B a FSM-C reagují na tento impulz současně. FSR-A je nezavislý a poskytuje data pro aktualizační funkce segmentů FSR-B a FSM-C. Nelineární kombinace hodnot uložených v segmentech FSR-A, získané použitím různých kombinací S-boxů, jsou použity pro aktualizační funkce všech částí systému. S-boxy S_A, S_B a S_D mají všechny dvaceti bitový vstup z FSR-A a poskytují čtyř bitový výstup. Kdežto s-boxy S_P a S_Q mají jako vstup 5 bitů z FSR-A a poskytují jedno bitový výstup. Keystrem je generován jako série dvou bitových slov. Na těchto slovech se podílí vnitřní stavy všech obvodů - FSR-A, FSR-B a FSM-C [2].

2.2.4. Bezpečnost

Bezpečnost tohoto algoritmu je zcela zásadní věc. Pokud by někdo našel účinnou metodu na prolomení šifry, tak by to mělo nedozírné ekonomické následky. Všechny placené televizní služby by rázem byly zbytečné. Každý by mohl sledovat libovolný zakódovaný program i bez dekodéru od poskytovatele signálu. Jelikož se na světě vyskytuje několik set miliónů DVB přijímačů, byl by to globální problém.

Asi nejznámějším útokem na CSA byl útok **Weinmann and Wirt.** Tento útok se zaměřuje na streamovou část algoritmu, přesněji na vnitřní stav šifry během autonomního stavu FSR-A. Složitost útoku by měla být menší než 2^{45}. Nicméně útok má jednu slabinu a ve skutečnosti je méně efektivní než útok hrubou silou (brute force attack). Tento útok probíhá ve třech fázích. První fáze odhaduje 53 bitů vnitřního stavu algoritmu (zahrnujíc FSR-A, FSM-C). Funkce Fc (kóduje výstup z FSR-B) je lineární kombinací neznámých bitů z FSR-B a výstupy této funkce jsou lineárně kombinovány do formy keystreamu, můžeme vytvořit soustavu rovnic, tato soustava řeší vztah mezi keystreamem a obsahem FSR-B. Složitost této fáze je 2^{53}. Druhá fáze řeší tyto rovnice pomocí Gaussovy eliminační metody. Složitost této fáze je v průměru $2^{17,7}$. Poslední část se kontroluje konzistentnost získaných dat. Pokud selže tak se odhadne nová hodnota a postup se opakuje. Ale když je původní odhad korektní, řešení rovnic je použito ke zjištění počátečního stavu a zjištění klíče. Složitost této části je zanedbatelná [2].

V reálných systémech je účinným způsobem zabezpečení vysílání klíče, které se každých pár sekund mění. Tím se zamezí tomu, že i kdyby se podařilo někomu získat klíč, tak mu bude za chvíli k ničemu. Tato změna je realizována tak, že údaje o klíči se přenášejí jako součást adaptačního pole přímo v TS paketech.

2.3. DVB

V předchozích kapitolách byla zmíněná zkratka DVB. Co je to vlastně DVB ? Projekt Digital Video Broadcasting je technologické mezinárodní konzorcium tvořené 250 členy z řad televizních společností, výrobců, síťových operátorů, softwarových vývojářů a regulačních orgánů ze 35 zemí z celého světa. Jeho cílem je vytvářet interoperabilní standardy pro globální poskytování mediálních služeb. Služby poskytované pomocí DVB specifikací jsou dostupné na všech kontinentech. V lednu 2011 bylo aktivních přes 500 miliónů DVB přijímačů. DVB porývá celou škálu kvality vysílání od televize s nízkým rozlišením LDTV přes standardní SDTV až po nejvyšší HDTV. Stejně tak pokrývá celou škálu kvalit zvuku - monofonní, stereofonní i prostorový zvuk 5.1 (Dolby Digital).

V první fázi bylo cílem snažení DVB poskytnout možnost digitálního televizního vysílání třemi základními distribučními sítěmi. Byly to standardy pro satelitní vysílání DVB-S, pozemní nebo také terestriální vysílání DVB-T a kabelové vysílání DVB-C. Novějším standardem je DVB-H (handheld), jak už anglický název napovídá je toto vysílání určeno pro

kapesní zařízení jako jsou mobilní telefony. Poslední velmi rychle se rozšiřující specifikací je DVP-IPTV, je určená pro přenos televizního signálu prostřednictvím počítačových sítí. Tyto standardy stručně zcharakterizuji v následujících podkapitolách.

2.3.1. DVB-T

DVB-T je nejrozšířenější ze standardů DVB. Do konce roku 2010 bylo celosvětově prodáno přes 200 miliónů přijímačů. Jeho hlavní zastoupení je v Evropě, Austrálii a jihovýchodní Asii. Je to technický standard, který specifikuje strukturu formování obrazu, kanálové kódování a modulaci pro digitální pozemní vysílání.

Je to velice flexibilní systém. Byl navrhnut pro poskytování široké škály služeb jako je: HDTV, vícekanálová SDTV. Signál má více možností příjmu: pevný, přenosný a mobilní. Vývoj musel počítat s velkými úskalími jako šum při přenosu, šířka pásma a rušení při všech krocích přenosu signálu.

DVB-T jako jiné běžně používané systémy pro pozemní vysíláni používá COFDM (kódovaný ortogonální multiplex s kmitočtovým dělením) modulaci. Tento typ modulace poskytuje velké množství nosných kmitočtů. Flexibilitu poskytuje díky množství nastavení: systémy 2k nebo 8k poskytující počet nosných frekvencí; 3 varianty modulace - QPSK, 16-QAM a 64-QAM; možnost pracovat v šířce pásma 6, 7 nebo 8MHz; 5 možností kódového poměru označovaného jako FEC (Forward Error Correction) a 4 ochranné intervaly. Toto vše poskytuje obrovské množství nastavení, ze kterých si poskytovatel služby může vybrat přesně to, které bude ideální pro jeho požadavky. Dalším charakteristickým znakem je hierarchické modulování - přenášení jedné modulace v druhé.

2.3.2. DVB-H

DVB-H je standard učený pro vysílání digitální televize pro mobilní zařízení, jako jsou mobilní telefony či PDA. Je to specifikace fyzické vrstvy umožňující přenášet data zapouzdřená v IP paketech prostřednictvím existujících terestriálních sítí. Úzce souvisí s DVB-T. Proč ale vytvářet nový standard, když DVB-T umožňuje příjem i na mobilních zařízeních? Hlavním důvodem je vytvoření mobilní TV, video streamu a stahování souborů na mobilních zařízeních s velkou úsporou energie pro přenos, jelikož všechny mobilní zařízení mají omezený zdroj energie.

DVB-H je rozšířením DVB-T s částečnou zpětnou kompatibilitou (např. může sdílet stejný multiplex s DVB-T). Je v něm použit systém zvaný multi-protokolové zapouzdření umožňující přenášet data síťových protokolů pomocí MPEG-2 transport streamu. Navíc oproti DVB-T přibyl systém nosných frekvencí 4k pro větší flexibilitu sítě.

Dalším důležitým aspektem je časové segmentování (Time-Slicing). Jeho princip spočívá v přenášení dat ve shlucích (burstech) tzn. jen v určitých časových intervalech. Data

přicházejí v několika sekundových intervalech. Když přijímač nepřijímá požadovaná data, je tuner neaktivní. Tím klesá spotřeba baterií. Jejich využití je tím pádem efektivnější. Spotřeba tuneru podporujícího Time-Slicing je o 95% nižší než u klasického DVB-T tuneru. Vysílač je však neustále aktivní. Další výhodou tohoto systému je schopnost že, v neaktivní době přijímače pro příjem dat dané služby, může přijímač vyhledávat vysílače sousedních buněk, které poskytují stejnou službu. Při přemístění do sousední buňky je pak přijímač snadno přeladěn.

2.3.3. DVB-S

DVB-S je digitální satelitní přenosový systém. Satelitní vysílání bylo prvním cílem organizace DVB. Standardy DVB tvoří základ většiny dnes používaných digitálních TV systémů. První satelitní systémy na světě byly spuštěny v roce 1994 právě na platformě DVB-S. Postupem času se stal nejpopulárnějším satelitním systémem. Na konci roku 2010 bylo celosvětově v provozu přes 100 miliónů DVB-S přijímačů. Jelikož je systém víc než 15 let starý, existuje již specifikace DVB-S2, která přináší lepší kanálové kódování, novější typy modulace a vylepšenou opravu chyb přenosu. To vše, spolu s novými technologiemi komprese obrazu, přináší prostor pro rozšíření satelitní HDTV. Jelikož je první verze systému na ústupu před druhou verzí, popíšu charakteristiku druhé generace.

DVB-S2, stejně jako první generace systému, je založen na QPSK modulaci společně s různými typy kanálového kódování a korekci chyb. Později byla implementovaná podpora 8PSK a 16QAM modulací. Nejzajímavějším aspektem DVB-S2 je ACM - Adaptivní kódování a modulace. To umožňuje měnit parametry vysílání při normálním provozu v závislosti na podmínkách přenosové cesty ke každému odběrateli [3].

2.3.4. DVB-C

Tento systém byl poprvé publikován v roce 1994 a postupně se stal nejrozšířenějším systémem digitální kabelové televize. Zajímavostí je, že systém DVB-C je taktéž integrován, v podobě fyzické vrstvy, v evropské verzi DOCSIS (Data Over Cable Service Interface Specification).

DVB-C může využívat pouze 1 transport stream. Systém nevyužívá ochranných intervalů. Využívá modulační schéma 16- až 256-QAM. Použitá modulace je typu QAM - pouze jedná nosná. Kódování a modulace je konstantní. V dnešní době, spolu s rostoucí poptávkou rozsahu služeb, většina kabelových poskytovatelů vylepšila svoje sítě na modulační schéma 256-QAM (maximální možná rychlost je pak 50 Mbps na 1 kanál). To poskytuje poskytovatelům velké možnosti v nabídce služeb. Mnoho operátorů poskytuje souběžně několik desítek analogových programů, stejně tak i digitálních a nové stále zařazují do svých nabídek.

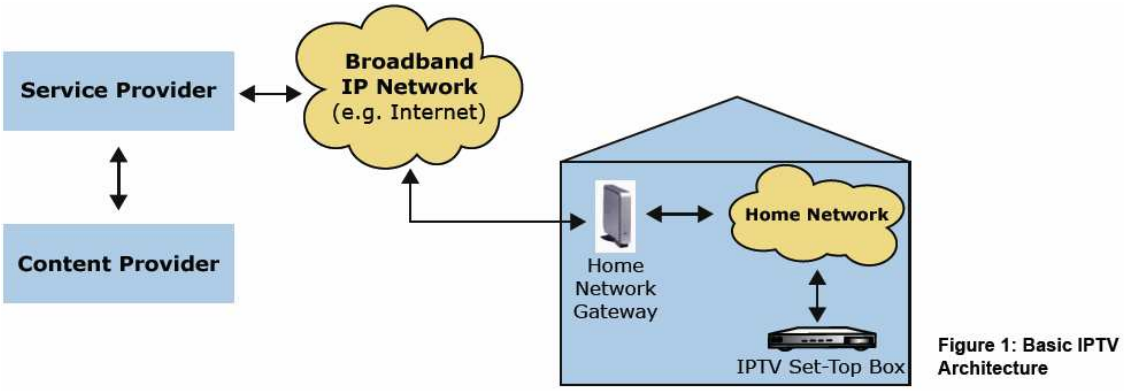
Jelikož požadavky stále rostou, objevila se na začátku roku 2010 specifikace DVB-C2. Přináší několik novinek oproti svému předchůdci. Schopnost přenášet několik transport

streamů. Variabilní kódování i modulace. Přechod na COFDM a tím zvětšení počtu nosných. Rozšíření schémat modulace až na 4096-QAM. Novinkou je taky použití ochranných intervalů [4].

2.3.5. DVB-IPTV

Tento standard se trochu liší od jiných DVB standardů. Není to jedna specifikace přímo určující principy a technologie přenosu, ale je to sada otevřených, technických specifikací vytvořených organizací DVB. Usnadňuje přenos digitálního televizního vysílání pomocí obousměrných, širokopásmových sítí. Má dvě hlavní části (obě pracují na IP sítích) - první část se zaměřuje na přenos prostřednictvím malých spravovaných sítí a druhá prostřednictvím internetu.

S příchodem vysokorychlostního, širokopásmového připojení rostly i požadavky na přenos služeb DVB prostřednictvím IP sítí. Přenos televizního signálu prostřednictvím IP sítí přináší mnoho problémů, zejména pokud je nutností zachovat IP služby sítě. Cílem DVB je definovat a vytvořit vhodné standardy pro přenos služeb prostřednictvím sítí tak, aby je bylo možno integrovat s jinými širokopásmovými službami sítě se zachováním maximální možné úrovně součinnosti s již existujícími DVB službami [5].



Obrázek 7.[Zdroj: www.dvb.org]

Obrázek 7. znázorňuje schéma DVB-IPTV přenosové cesty. Počáteční fáze se zaměřovala na rozhraní mezi IPTV set-top-boxem (STB) a domácí sítí (home network). Bylo nutné vytvořit vhodnou definici automatizovaného připojení a konfiguraci set-top-boxu připojeného do IP sítě.

3. Implementace

3.1. Současný stav

Jak jsem již zmiňoval v úvodu, dosud neexistuje výkonově nenáročná aplikace pro kódování MPEG-2 Transport Streamu. Existuje VLC Media Player organizace VideoLAN, jehož jednou z mnohá funkcí je právě kódování transportního streamu algoritmem CSA. Nicméně je náročný jak pro uživatele (množství různých nastavení může zmást uživatele), tak pro počítač - vyvíjí velkou zátěž na procesor a paměť.

3.2. Specifikace požadavků

Program by měl mít tyto vlastnosti:

- **Nenáročnost** - program by neměl příliš zatěžovat počítač.
- **Instalace -** samozřejmostí je jednoduchá instalace pomocí MAKE.
- **Uživatelská přívětivost** - Jednoduchý HELP, přehledné nastavování parametrů programu (Nastavení vstupní a výstupní adresy a portu, volba klíče kódování, možnost volby, které PID se budou kódovat).

3.3. Analýza

Již ze samotného zadání je zřejmé, že program bude pracovat se sítí. Bude fungovat jako proxy server. Data bude zachytávat na vstupním socketu. Po jejich zpracování je odešle zpět do sítě opět pomocí socketu.

Jelikož komunikace bude probíhat pomocí datagramového protokolu UDP, nebude nejspíš možno jednoduše nastavit velký počet dat, která se mají najednou ze socketu přijmout. Existují dvě varianty řešení tohoto problému: statické dvourozměrné pole nebo dynamické pole, která budou sloužit jako vstupní buffer pro data.

Klíčovou části programu bude samotné zpracovávání paketů transportního toku. Jsou zde dvě hlavní možnosti. Mít uloženou pouze jednu kopii dat, ve které zakódujeme potřebné pakety a odešleme do sítě. Druhá varianta spravuje dvě kopie dat. V první se zakódují všechny pakety a ve druhé se nezmění nic. Následně při odesílání zpět se na základě ID paketu rozhodne, ze které kopie se paket odešle.

3.4. Knihovna LIBDVBCSA

Libdvbcsa je volně rozšířitelná a přenositelná implementace DVB CSA. Poskytuje funkce pro kódování i dekódování streamu. Má tyto vlastnosti:

- **Přenositelnost.** Knihovna byla úspěšně testována na různých procesorech s 32, 64 i 128 bitovou architekturou a řazení bytů little-endian a big-endian.

- **Výkon.** Knihovna poskytuje 2 varianty kódování. Klasickou jednoduchou, nazývanou taky sériová a rychlejší paralelní verzi. Ta může využít výhody následujících sad instrukcí: MMX, SSE nebo Altivec. Dle dokumentace může paralelní implementace zvládat stream až 300 Mbps.

- **Open source.** Knihovna je vydána pod General Public License, která zajišťuje, že je zdarma k dispozici a bude použita pouze ve volně používaných produktech.

- **Jednoduchost.** API má pouze 5 funkcí pro klasickou implementaci a 6 pro paralelní verzi.

3.4.1. Klasická verze

Tato klasická implementace může zpracovat pouze 1 datový paket na 1 funkční volání. Z toho vyplývá horší efektivita. Je to nejpomalejší verze implementace. Proto je vhodnější ji používat pouze na slabších strojích, kde by rozdíl v výkonu oproti paralelní verzi nebyl tak znazelný. Je proto používaná pouze, když není k dispozici větší skupina paketů najednou neboli u pomalých streamů. Dokumentace udává, že zvládá streamy s bitratem od 20 Mbps do 50 Mbps. Velkou výhodou této varianty je jednoduchost programu. Data stačí pouze přijmout, zakódovat a po jednotlivých paketech zase odeslat. Nemusíme tedy řešit jak naplnit pole a ukládat tyto data. Uvedu stručný popis použití.

```
void dvbcsa_encrypt (const struct dvbcsa_key_s *key, unsigned char *data, unsigned int len);
```

Zadáváme ukazatel na přijatá užitečná data, nikoliv na celý paket. Dále délku samotných dat v tomto paketu. Funkce vyžaduje na vstupu klíč, dle kterého se budou data kódovat. Vytvoříme si proměnnou s danou strukturou, které následně nastavíme hodnotu:

```
unsigned char cw[8] = "testtest";
struct dvbcsa_bs_key_s *key = dvbcsa_bs_key_alloc();
dvbcsa_bs_key_set(cw, key);
```

3.4.2. Paralelní verze

Paralelní implementace je rychlejší než klasická, ale datové pakety musí být seskupeny. Z toho vyplývá složitější implementace v programu. Pro volání metod pro kódování nebo dekódování musí být skupina paketů uložená v poli datové struktury dvbcsa_bs_batch_s.

```
struct dvbcsa_bs_batch_s
{
        unsigned char       *data;   //ukazatel na data
        unsigned int        len;     //délka dat
}
```

Pole nesmí být větší než hodnota, kterou vrací funkce unsigned int dvbcsa_bs_batch_size(void); toto číslo je závislé na architektuře procesoru. Nesmíme ovšem zapomínat na ukončení datového pole hodnotou NULL, touto hodnotou se ukončuje kódování. Zakódování pole s menším počtem paketů zabere stejný čas jako u pole o maximální délce.

```
void dvbcsa_bs_encrypt(const struct dvbcsa_bs_key_s *key, const struct dvbcsa_bs_batch_s *pcks, unsigned int maxlen);
```

Prvním potřebným parametrem pro správný běh kódovací funkce je osmi bytový klíč. Podle tohoto klíče se data zakódují. Vytvoření struktury a nastavení hodnoty klíče v této verzi je prováděno obdobně. Jediný rozdíl je v názvu volaných funkcí - struct dvbcsa_bs_key_s * dvbcsa_bs_key_alloc(); Pro vytvoření klíče a void dvbcsa_bs_key_set(const dvbcsa_cw_t cw, struct dvbcsa_bs_key_s *key); pro jeho nastavení na danou hodnotu. Po klíči musí následovat samotná struktura s daty.

Dalším vstupním parametrem metody pro kódování paralelním způsobem je maximální délka dat. Toto číslo musí být násobkem 8. Přímo ovlivňuje chod algoritmu, řídí počet cyklů. Mělo by být pokud možno co nejmenší. Pro pakety MPEG transportního toku je tato hodnota rovna číslu 184. Data přesahující tuto hodnotu se nezpracují celá, nezakódují se ty byty, které přesahují tuto hranici.

Je evidentní, že tato možnost bude složitější na samotnou implementaci v programu. Nicméně by měla být výkonnější než klasická verze. Uvidíme zda nárůst výkonu bude natolik velký aby se tím vyrovnala složitost implementace.

3.5. Popis implementace

V kapitole 3.3 Analýza jsem se věnoval problému implementace pouze obecně. V této části bych se rád věnoval přímo mojí implementaci i s ukázkami kódu z mého programu. Uvedu taky zdůvodnění proč jsem zvolil danou variantu řešení.

3.5.1. Konstanty a knihovny

Kromě standardních knihoven pro práci s řetězci (stringy), sockety atd. jsem musel importovat rovněž knihovnu LIBDVBCSA. Pro obě verze implementace se importuje shodná knihovna, nebylo tedy nutno import rozlišovat klasickou nebo paralelní implementaci. Protože knihovna LIBDVBCSA je napsána v jazyce C, nikoli C++, tak musíme import knihovny definovat v klauzuli extern "C" . Bez této klauzule je možno importovat pouze knihovny jazyka C, které jsou poskytovány systémem.

```
11      extern "C" {
12              #include <dvbcsa/dvbcsa.h>
13      }
```

Důležitou věcí pro přehlednost a samotný chod programu jsou nadefinované konstanty, které jsou v programu využívány. Hodnota TS_SIZE určuje velikost jednoho paketu v bytech, taky ji v programu využívám při nastavování ukazatelů na začátky paketů. Konstanta PACKETS určuje počet paketů v jednom datagramu. Konstanta BLOCKS určuje počet datagramů, které bude program přijímat při jednom cyklu zpracování paketů.

```
17      #define TS_SIZE 188
18      #define PACKETS 7
19      #define BLOCKS 30
```

3.5.2. Plnění bufferu a kódování

Jelikož sockety neumožňují přijetí více datagramů pomocí jednoho funkčního volání funkce recvfrom, rozhodl jsem se to vyřešit pomocí dvourozměrného staticky alokovaného pole. Toto pole si můžeme představit jako jednorozměrné pole přijatých datagramů.

```
203     unsigned char buf[BLOCKS][PACKETS*TS_SIZE];
```

Pole pro buffer je alokované pomocí konstant. Pro staticky alokované dvourozměrné pole jsem se rozhodl z důvodu menší náročnosti na výpočetní schopnosti procesoru. Dynamické pole by bylo složitější na realizaci a zvětšování jeho velikosti by mohlo zbytečně zatěžovat procesor.

```
214  for(int i=0;i<BLOCKS;i++)
216     result = recvfrom(sockfdi, buf[i], sizeof(buf[i]), 0,(sockaddr *)&addressi, &leni);
```

Konstantu BLOCKS nevyužívám pouze pro alokování pole bufferu, ale i jako řídící hodnotu pro přijímací cyklus nebo odesílací (data odesílám stejným principem jako příjmám pomocí smyčky FOR).

Dalším využitím konstant BLOCKS a PACKETS je procházení přijatými daty. Jsou využívány jako řídící hodnoty počtu proběhnutí vnořených cyklů FOR. Tento průchod realizuji pomocí vnořených cyklů. V prvním cyklu program prochází jednotlivými datagramy. Ve druhém (vnořeném) cyklu program prochází a zpracovává jednotlivé pakety.

Jelikož všechny funkce pro kódování dat vyžadují ukazatele na zpracovávaná data, je nutno si vytvořit ukazatele na začátky přijatých paketů. Opět jsem se rozhodoval mezi dvojicí možných řešení. První možností (mnou nezvolenou) bylo vytvoření pole ukazatelů na místa kde se v bufferu nacházejí začátky jednotlivých paketů. Tuto možnost jsem nezvolil z důvodu paměťové náročnosti. S rostoucí velikostí buffer by rostlo i pole ukazatelů. Druhou možností, pro kterou jsem se rozhodl, je nastavení hodnoty ukazatele před každým zpracováním jednoho konkrétního paketu. Toto řešení šetří paměť, protože využívá pouze jednu proměnnou.

```
221/223    buf_p = &buf[i][j*TS_SIZE];
```

Takto vytvořený ukazatel na paket stále ještě nejde použít pro funkci kódování. Funkce, jak jsem již zdůraznil v kapitole 3.4, pracují s ukazatelem na samotná data, nikoli na celý paket. Pokud se v paketu nevyskytuje adaptační pole, tak je řešení jednoduché - pouze ukazatel zvýšíme o 4 (délku záhlaví paketu). Pokud však paket obsahuje adaptační pole, musel sem v programu zjistit jeho délku a zvýšit jí o 1 (z důvodu, že délka adaptačního pole nezohledňuje byte ve kterém je uložena) a o tuto hodnotu zvýšit hodnotu ukazatele.

Posledním důležitým argumentem kódovací funkce je délka dat určených k zakódování. Princip je obdobný jako u nastavování ukazatele. V případě absence adaptačního pole se délka nastaví na 184. Pokud paket pole obsahuje, vypočítá se délka odečtením délky pole inkrementována o 1 od maximální délky pole.

```
240/243    len = len - getAFL(buf_p) -1;
241/244    pointer = pointer + getAFL(buf_p) + 1;
```

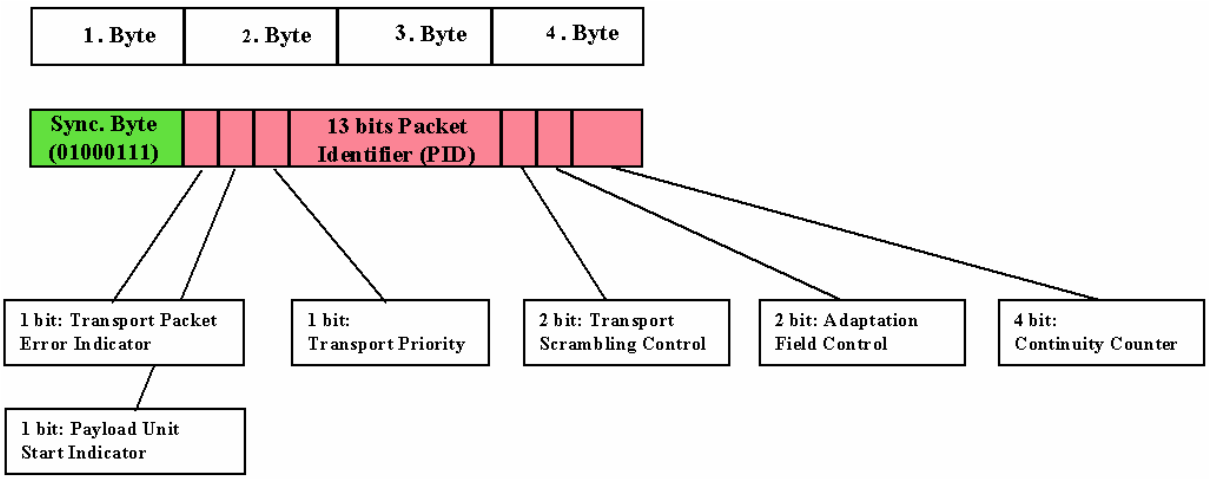
Všechny výše popisované části jsou společné pro obě varianty kódování. Při řešení klasické implementace nebylo nutné zajistit žádné další pomocné konstrukce. Pouze se na základě ID paketu a řídící hodnoty adaptačního pole rozhodne, zda se má daný paket kódovat.

Paralelní verze je komplikovanější. Na začátku jsem se musel rozhodnout zda budu kódovat vždy maximální možný počet paketů v 1 funkčním volání nebo poslední volání kódovací funkce proběhne s menším počtem paketů, což by mělo za následek drobné snížení výkonu. První varianta přináší mnoho úskalí - při případném výpadku kódovaného streamu (pokud by program přijímal celý multiplex) by vstupní buffer neustále rostl. Hrozilo by i přetečení velikosti operační paměti. Musel bych přidat kontrolu maximální velikosti bufferu, pokud by byl stream pomalý, program by způsoboval zpoždění při přenosu. Tato varianta by navíc využívala dynamického pole, jehož nevyužívání jsem zdůvodnil v kapitole 3.3.

Rozhodl jsem se využít druhou variantu. Statický buffer se naplní daty, které se následně pomocí vnořených cyklů projdou a na základě ID paketů se rozhodne, zda se daný paket má kódovat. Pokud ano, začne se naplňovat struktura dat, která se předává kódovací funkci. Když se naplní, zakóduje se. Pokud po zpracování celého bufferu obsahuje nějaké pakety, tak ty se zakódují (ovšem se již nekóduje maximální možný počet paketů - snížení výkonu celého systému). Se zvětšováním bufferu roste i efektivita programu.

3.5.3. Funkce pro práci se záhlavím

Bylo by neefektivní přenášet jednobitové řídící pole v jednom bytu (záhlaví by narostlo ze 4 bytů na 8 bytů), tudíž se řídící pole spojují do souvislého toku, který se rozdělí do jednotlivých bytů. Na obrázku číslo 8 je znázorněno rozložení jednotlivých řídících polí záhlaví do bytů, tak jak jsou přenášeny v reálných systémech.



Obrázek 8.[Zdroj: www.une.edu.ve]

Je zřejmé, že k těmto datům nemůžeme přistupovat přímo. Pro čtení a upravování záznamů v záhlaví budeme potřebovat bitové operace a bitové masky. V programu využiji zejména funkce pro 4 úkony. Zjištění identifikačního čísla paketu (PID), zjištění přítomnosti adaptačního pole, zjištění délky adaptačního pole a nastavení příznakových bitů kódování dat.

- Funkce int getPID(unsigned char *pom) - Tato funkce, jak již její název napovídá, vrací PacketID daného paketu. Jako jediná z pomocných funkcí pracuje se dvěma byty. Na obrázku 6 vidíme , že PID je rozděleno do dvou bytů. Horních pět bitů je uloženo ve druhém bytu a dolních osm bitů ve třetím bytu. Třetí byte nevyžaduje žádnou úpravu. V druhém bytu nejdříve odstraním horní tři bity (pomocí logického součinu s hodnotou 0x1f) a následně pomocí bitového posuvu těchto 5 zbylých bitů posunu o osm doleva. Výsledek bitového posuvu pak pomocí funkce OR spojím se spodní částí, která je uložená ve třetím bytu.

- Funkce int getAFE(unsigned char *pom) - Adaptation Field Exist - vrací hodnotu řídícího pole Adaptation Field Control (indikuje existenci adaptačního pole). Její funkce je jednoduchá. Pouze odmaskuji nepotřebnou část čtvrtého bytu (pomocí funkce AND a hodnoty 0x30) a výsledek se vrátí.
- Funkce unsigned int getAFL(unsigned char *pom) - Adaptation field Length - používá se pouze v případě, že v daném paketu je obsaženo adaptační pole. Princip funkce je velmi jednoduchý. Pouze se vrátí hodnota pátého bytu paketu.
- Procedura void setSCR(unsigned char *pom) - je jedinou procedurou pro obsluhu záhlaví paketu. Nastavuje hodnotu řídícího pole Scrambling control na hodnotu 11 a to pomocí logického součtu čtvrtého bytu paketu a hodnoty 0xC0.

3.6. Funkčnost

Zatím jsem popisoval pouze implementaci, ale otázkou je zda program vůbec pracuje jak má. . Výpis programu z terminálu (realizováno pomocí funkce HexDump), pro přehlednost budu vždy uvádět jako první nekódovaný paket.

Pro všechny níže uvedené výpisy z terminálu platí následující barevné rozlišení: Zelená- synchronizační byte, červená - zhlaví paketu, šedá - užitečná data (payload), růžová je vyhrazená pro délku adaptačního pole a modrá pro adaptační pole.

V prvním výpise je uveden fragment "klasického" paketu, bez adaptačního pole. Kódováním neprochází pouze opravdu záhlaví (první čtyři byty paketu).

```
0000    47 00 45 15 bc e2 50 80 0a bf f8 6f d9 d5 64 9f    G.E...P....o..d.
0010    ff 52 1b 58 b3 7b 64 e1 88 2c c0 e8 31 95 f9 84    .R.X.{d..,..1...
0020    58 04 05 81 54 93 03 00 cb 3e 74 64 a0 68 c1 82     X...T....>td.h..

0000    47 00 45 d5 98 33 e2 2c dd 2a e5 75 23 3f 73 41    G.E..3.,.*.u#?sA
0010    01 df 70 c6 54 21 f9 9c d4 82 cc bf d7 42 af c0    ..p.T!.......B..
0020    b9 1d cf 0f ae 35 d1 72 a5 07 b5 b8 f6 7b 7d ff     .....5.r.....{}.
```

Druhý úryvek z výpisu terminálu znázorňuje paket, který obsahuje adaptační pole o délce sedmi bytů. V tomto adaptačním poli se přenáší pomocné řídící informace (PCR - vysvětleno v kapitole 2.1.3.).

```
0000    47 00 45 33 07 10 1d 86 b0 97 fe 00 a9 a3 ad a0    G.E3............
0010    83 be 5b 0c be 77 5b 4c 65 63 d1 42 64 df 4d b7    ..[..w[Lec.Bd.M.
0020    d8 cc 9f 88 9c ea b6 cb 1c ff 17 7d 46 1d dd dd    ...........}F...

0000    47 00 45 f3 07 10 1d 86 b0 97 fe 00 b3 9f d8 f8    G.E.............
0010    db 2d 3b f4 1f 49 1e b7 2a f4 9f fa 79 29 4e 7f    .-;..I..*...y)N.
```

```
0020    8a 37 a9 a4 5d 6f 6b 90 58 6d a0 bc fb fa ff ee    .7..]ok.Xm......
```

Poslední výpis zobrazuje paket, který obsahuje adaptační pole, které slouží pouze jako doplnění paketu daty tak, aby splňoval požadovanou délku 188 bytů. Je zde dobře vidět, že délka paketu je doplňována opravdu hodnotami 0xff jak bylo již dříve uvedeno v jedné z předchozích kapitol.

```
0000    47 00 45 36 11 00 ff ff ff ff ff ff ff ff ff ff    G.E6............
0010    ff ff ff ff ff ff ef e8 7a 9e 6b bf 8b d5 73 fe    ........z.k...s.
0020    5b c4 8b fc 8e ab f8 cd de ae 86 51 5e 3b dd 99    [.........Q^;..

0000    47 00 45 f6 11 00 ff ff ff ff ff ff ff ff ff ff    G.E.............
0010    ff ff ff ff ff ff 9f 87 8d ea dc 9a f2 43 f1 05    .............C..
0020    06 17 f1 55 f7 8d ca e7 b0 f3 db 9a b6 dc 3f 4e    ...U..........?N
```

Při výpisech paketů s adaptačním polem si můžeme všimnout, že na konci bloku řídících pomocných dat se nachází byte s hodnotou 0x00, tímto se demultiplexeru oznamuje, že končí řídící data a začínají pouze doplňovací byty s hodnotou 0xFF.

3.7. Srovnání výkonu obou verzí algoritmu

Nejzajímavější částí je bezesporu srovnání výkonu. Za ideálních podmínek by měla být paralelní implementace až 4x rychlejší, ale v praxi není nikdy nic ideální. Následující testy ukážou jak. Toto číslo je ale závislé na mnoha jiných parametrech (rychlost vstupního streamu, velikost vstupního bufferu).

Testy a porovnávání jsem prováděl na svém lokálním stroji, na kterém jsem pomocí VMWare Playeru emuloval OS Linux Ubuntu 9.10. Stream jsem vytvářel pomocí VLC Media player. Rychlost streamu se pohybovala okolo 1946 kbps. Pro rovnost podmínek jsem streamoval stejný videoklip v délce sedmi minut a čtyřiceti tří sekund. Klíč pro kódování byl 0xFFFFFFFF. Všechny procesorové časy jsem měřil pomocí utility TIME v OS Linux. Jako první jsem testoval paralelní verzi.
Uvádím výpis z linuxového terminálu:
Paralelní algoritmus při kódování video i audio paketů , velikost vstupního bufferu 210 paketů.

```
real    7m47.895s
user    0m4.144s
sys     0m9.027s
```

Následovala klasická verze se vstupním bufferem o velikosti 7 paketů.

```
real    7m51.579s
user    0m49.259s
```

```
sys     1m24.301s
```

Tyto naměřené hodnoty byly bezesporu zajímavé. Ovšem již ne očekávané při čase zobrazujícím dobu běhu systémových knihoven. Nějakou dobu mi trvalo najít skutečnost proč tomu tak je. Měl jsem pár možností, které to mohly způsobovat. Naštěstí to byla jedna z prvních variant, které mně napadly. Tyto zajímavé výsledky byly různou velikostí vstupních bufferů. Program s menším bufferem častěji přijímal a odesílal menší objem dat. V případě většího bufferu se data přijímala a odesílala ve větších shlucích, tudíž operační systém si mohl tyto operace optimalizovat, aby byl výsledek co nejefektivnější. Po vyrovnání velikosti vyrovnávacích pamětí jsem již dostal očekávané výsledky.

```
real    7m49.301s
user    0m49.029s
sys     0m9.541s
```

Až na tento rozdíl v čase běhu systémových funkcí jsou výsledky očekávané. Paralelní implementace dokonce vykazuje lepší výkon, než jsem očekával. Dle dokumentace je schopná zvládat až čtyřikrát mohutnější streamy, tudíž i výkon by měl být stejně rozdílný. Mé výsledky ukazují, že může být rychlejší téměř desetkrát. Tento rozdíl ve výkonu se jistě vyplatí, pokud vezmeme do úvahy složitější implementaci oproti klasické verzi.

Všechna měření probíhala na virtuálním stroji s parametry 1024 MB RAM a 1-jádrový procesor s frekvencí 1GHz. Ovšem měření na 1 stroji, kde se současně stream vysílá i přijímá, není optimální. Současný provoz vysílacího a kódovacího programu může ovlivňovat naměřené výsledky. Oba programy jsem proto znovu otestoval v reálné simulaci provozu na stanicích v učebně PorD403 na VŠB-TUO. Počítače mají následující konfiguraci: Intel Core2duo E7500 2,93GHz a 4GB RAM.

Paralelní implementace při kódování video paketů , velikost vstupního bufferu 210 paketů.

```
real    7m48.727s
user    0m10.429s
sys     0m0.584s
```

Klasická implementace, vstupní buffer 210 paketů.

```
real    7m58.356s
user    0m38.574s
sys     0m0.588s
```

Pro porovnání s již existujícím nástrojem uvedu časy, které dosahoval program VLCMediaPlayer. Pro objektivní porovnání jsem měření prováděl při stejných podmínkách jako při měření mých programů.

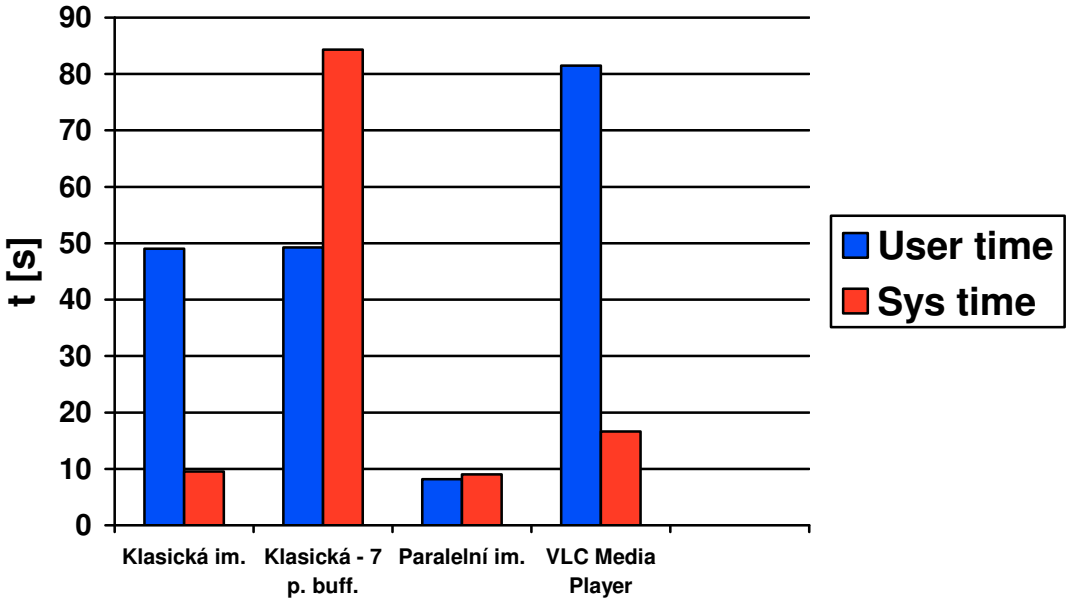
Časy na emulovaném systému:

real	7m48.307s
user	1m21.497s
sys	0m16.601s

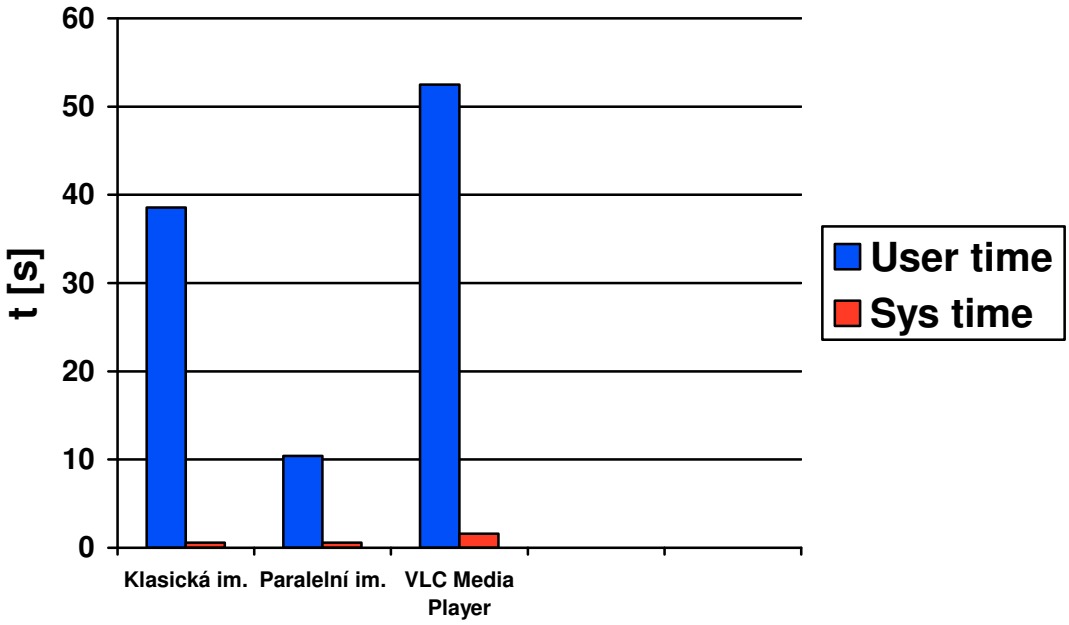
Časy na školním stroji:

real	7m45.984s
user	0m52.491s
sys	0m1.612s

Graf 1 - zobrazuje hodnoty naměřené na mém emulovaném systému.



Graf 2 - zobrazuje hodnoty naměřené na školních strojích.



Obě předchozí varianty měření pracovaly s relativně slabým streamem - okolo 2Mbps. Pro praktičtější údaje jsem provedl ještě měření doby kódování High Definition (HD) streamu. Tyto hodnoty budou poskytovat praktičtější porovnání, jelikož streamy se neustále zrychlují.

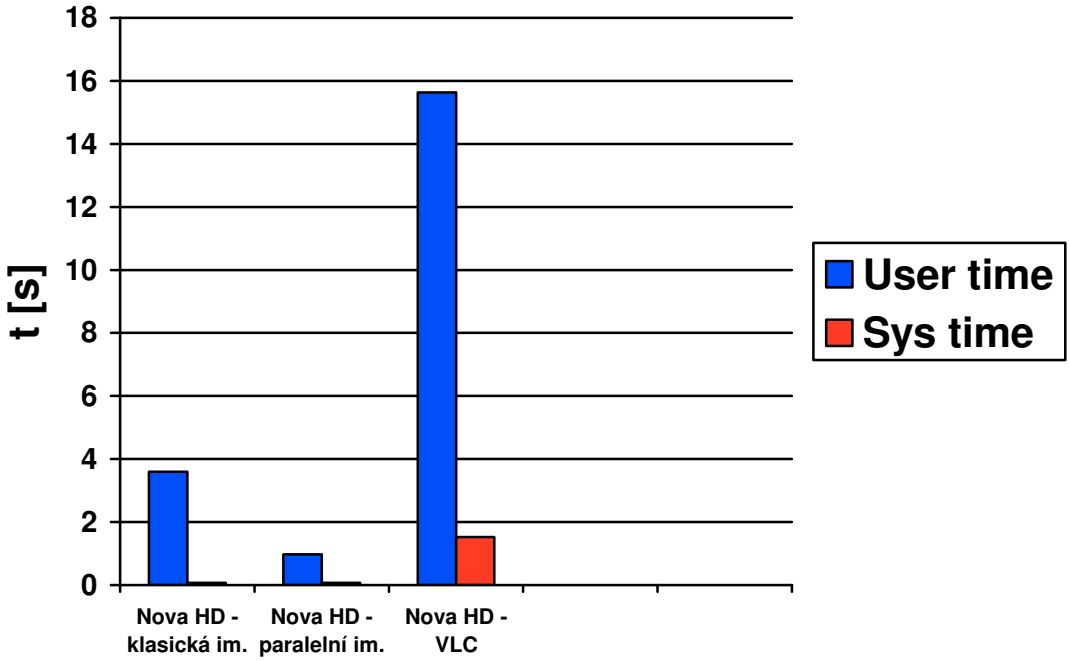
Naměřené hodnoty (v pořadí paralelní implementace, klasická implementace a VLC):

```
real    1m12.904s
user    0m0.978s
sys     0m0.072s

real    1m11.056s
user    0m03.600s
sys     0m0.052s

real    1m8.464s
user    0m15.629s
sys     0m1.528s
```

Graf 3 - kódování HD streamu.



4. Závěr

Tato práce byla velmi poučná a zajímavá a to hned z několika důvodů. Digitální televizní vysílání se čím dál tím víc dostává do podvědomí lidí, ale osobně jsem s ním neměl moc zkušeností a ani tušení na jakém principu pracuje. Při zpracovávání teorie MPEG-TS jsem se dozvěděl nejenom základní věci pro vytvoření programu, ale i jiné důležité informace o principu přenosu, které můžu využít později v praxi.

Při vývoji programu se ukázalo, že ne vždy je vše ideální a taky, že někdy se vyskytne problém tam, kde by zákonitě být neměl. Nejlepším příkladem při vývoji mé aplikace byla chyba, která se mi vyskytla při kódování. Dlouho jsem se jí snažil objevit. Podle výpisu zpracovávaných dat se vše jevilo v pořádku. Řešení bylo zdánlivě jednoduché, ale o to méně pochopitelné. Při vytváření kódovacího klíče stačilo prohodit dva řádky, konkrétně vytvoření řetězce kódovacího slova a vytvoření struktury, která je použitá jako vstupní argument kódovací funkce. Jako první se musel vytvořit řetězec znaků a až potom struktura pro klíč. Snažil jsem se následně vypátrat proč tomu tak bylo, ale nenašel jsem žádné zdůvodnění. I v oficiální dokumentaci je uvedené pořadí, které na mém stroji nefungovalo.

Testy prokázaly, že aplikace splnila očekávání - je méně náročná než dosud existující programové nástroje pro kódování streamů. Práce s použitím mé aplikace je rovněž jednodušší než práce s dosud existujícími programy, které odvedou stejnou službu.

Testování jsem prováděl při více konfiguracích. Program pracoval spolehlivě na různých strojích a při kódování různě mohutných streamů. Zajímavé bylo rovněž sledovat vliv různě velkých vstupních bufferů na výkon programu.

Z výsledků je patrné, že paralelní implementace přináší opravdu velký nárůst výkonu na obou testovaných systémech. Na mém emulovaném systému je tento nárůst výkonu patrnější, nicméně za průkaznější hodnoty bych uvažoval ty, které jsou naměřené na školních strojích - na školním systému je paralelní implementace téměř 4x rychlejší. Moje implementace jsou efektivnější než již zavedené nástroje. To bylo i hlavním cílem práce, který se podařilo splnit.

Literatura

[1] LEGÍŇ, Martin. Televizní technika DVB-T. Praha : BEN, 2007. Multilexování v DVB-T, s. 78. ISBN 978-80-7300-204-3

[2] SIMPSON, Leonie; HENRICKSEN, Matt; YAP, Wun-She. Specification of the CSA-SC. In Improved cryptanalysis of the Common Scambling Algorithm Stream Cipher [online]. Brisbane, Australia : [s.n.], 2009 [cit. 2011-04-22]. Dostupné z WWW: <http://eprints.qut.edu.au/>

[3] Fact Sheet. In DVB Fact Sheet - September 2010 2nd Generation Satellite [online]. [s.l.] : [s.n.], 2010 [cit. 2011-04-19]. Dostupné z WWW: <www.dvb.org>.

[4] Fact Sheet. In DVB Fact Sheet - September 2010 2nd Generation Cable [online]. [s.l.] : [s.n.], 2010 [cit. 2011-04-19]. Dostupné z WWW: <www.dvb.org>

[5] Fact Sheet. In DVB Fact Sheet - September 2010 Internet Protocol TV [online]. [s.l.] : [s.n.], 2010 [cit. 2011-05-03]. Dostupné z WWW: <www.dvb.org>

Obsah CD

Složka	Obsah
/src	Zdrojové kódy, Makefile a README
/text	PDF s textem práce